NASA
Technical
Paper
2454
C.1
July 1985

Intersection of Three-Dimensional Geometric Surfaces

Vicki K. Crisp,
John J. Rehder,
and James L. Schwing

NASA

NASA Technical Paper 2454

1985

Intersection of Three-Dimensional Geometric Surfaces

Vicki K. Crisp
Kentron International, Inc.
Hampton, Virginia

John J. Rehder
Langley Research Center
Hampton, Virginia

James L. Schwing
Old Dominion University
Norfolk, Virginia

NASA
National Aeronautics and Space Administration

Scientific and Technical Information Branch

Summary

Calculating the line of intersection between two three-dimensional objects and using the information to generate a third object is a key element in a geometry development system. Techniques are presented for the generation of three-dimensional objects, the calculation of a line of intersection between two objects, and the construction of a resultant third object. The objects are closed surfaces consisting of adjacent bicubic parametric patches using Bézier basis functions.

The intersection determination involves subdividing the patches that make up the objects until they are approximately planar and then calculating the intersection between planes. The resulting straight-line segments are connected to form the curve of intersection. The polygons in the neighborhood of the intersection are reconstructed and put back into the Bézier representation. A third object can be generated using various combinations of the original two. Several examples are presented.

Special cases and problems were encountered, and the method for handling them is discussed. The special cases and problems included intersection of patch edges, gaps between adjacent patches because of unequal subdivision, holes, or islands within patches, and computer round-off error.

Introduction

In recent years, considerable growth has occurred in the application of computer-aided-design (CAD) systems. These systems are being used in the design process from the conceptual and preliminary levels through the final manufacturing stages. A key element in all CAD systems is the description of the geometry of the object being designed. A three-dimensional (3-D) geometrical description can be used to determine weights and assess clearances as well as to provide the designer with the essential visualization of the object being designed. For aerospace vehicles, the geometry is applied to aerodynamic and heating prediction methods to initially design the shape, structural concept, and thermal protection system.

At the Langley Research Center, CAD techniques have been applied to the conceptual and preliminary design of advanced space transportation concepts. The Aerospace Vehicle Interactive Design (AVID) system has been developed for that purpose (ref. 1). An important part of AVID is the geometry development system, a full 3-D modeller, based on bicubic Bézier patches, that allows a designer to describe complex vehicle shapes in a natural way. The designer is able to create 3-D objects by modifying primitive objects, by extending two-dimensional (2-D) shapes by revolving or lofting, and by merging two objects to form one. An example of object merging is the joining of a wing and a fuselage to form one continuous object by eliminating the portions of each object that lie within the other.

Forming a composite object by merging two objects has always been a problem for geometry modellers. Two distinct problems must be solved. First, the curve of intersection between the objects must be determined. Then, using the intersection information, a resultant object must be formed that is mathematically consistent with and geometrically faithful to the original objects.

Recent efforts to calculate the curve of intersection between curved surfaces have used the technique of subdividing the surfaces into planar polygons and determining the intersection between planes. The intersection curve can be generated by connecting the line segments produced by the planar intersections. In particular, the current effort is based principally on previous work by Carlson (ref. 2). Newer techniques for intersection tests and calculations were applied within Carlson's overall algorithm; the most significant technique was a subdivision method from Lane and Riesenfeld (ref. 3).

The most difficult part of merging objects is the construction of the resultant object once the intersection has been identified. So far, no method has been found that retains the original surface properties such as curvature. The current study presents a technique for converting the irregularly shaped polygons generated by the intersection back into the Bézier representation. Although the original curvature of the surfaces is still lost in the neighborhood of the intersection, the resultant objects created by combining the objects in different ways are faithful enough to be used in a preliminary design system.

This paper describes a technique for merging objects. The mathematical basis for defining the objects as closed surfaces consisting of adjacent bicubic patches is presented. In particular, a method for generating bodies of revolution is described. Techniques for determining the space curve of intersection between the

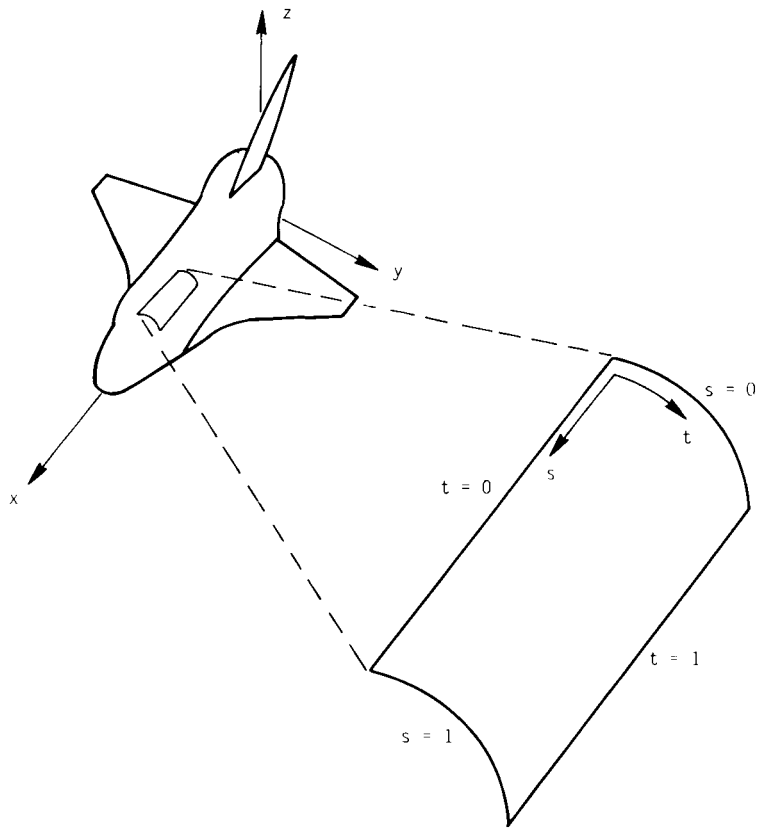


Figure 1. Coordinate system and patch representation.

two objects and the portion of each object which lies inside the other are presented. Finally, the method of developing the resultant object for the desired geometric combination is described. Computer algorithms and programs for constructing, merging, and displaying 3-D objects are discussed.

3-D Object Generation

Mathematical Basis

The coordinate system and the patch representation for this study are illustrated in figure 1. A right-handed Cartesian coordinate system (x, y, z) was used. The surfaces under consideration are comprised of adjacent four-sided bicubic patches. The surface equation for the patch is a cubic polynomial in two parametric variables (s and t) that range in value from 0 to 1. The four sides of the patch are cubic space curves formed when $s = 0$, $s = 1$, $t = 0$, and $t = 1$.

The form of $x(s,t)$ is as follows (ref. 4):

$$x(s,t) = a_{11}s^3t^3 + a_{12}s^3t^2 + a_{13}s^3t + a_{14}s^3 + a_{21}s^2t^3 + a_{22}s^2t^2 + a_{23}s^2t + a_{24}s^2 + a_{31}st^3 + a_{32}st^2 + a_{33}st + a_{34}s + a_{41}t^3 + a_{42}t^2 + a_{43}t + a_{44}$$

This equation is more conveniently written in matrix notation as

$$x(s,t) = SC_xT^T$$

where $S = [s^3\ s^2\ s\ 1]$, $T = [t^3\ t^2\ t\ 1]$, and C_x is the polynomial-coefficient matrix. There are similar equations for $y(s,t)$ and $z(s,t)$.

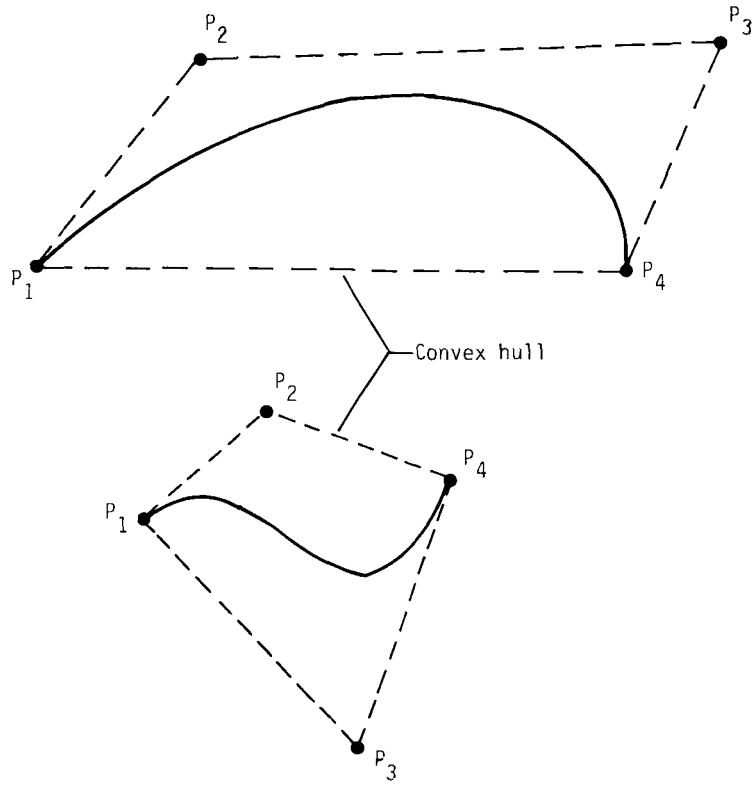


Figure 2. Examples of 2-D Bézier cubic curves.

There are several possible mathematical bases for the coefficient matrix. The particular form of cubic equation used in the present work is the Bézier polynomial. (See ref. 5.) In two dimensions, a Bézier curve is defined by four control points. (See fig. 2.) The two endpoints and the tangent vectors at the endpoints are specified. The curve can be modified by direct manipulation of the four points. These points form a polygon, called the convex hull, which completely bounds the curve. The parametric cubic equation for the curve is $x(t) = TM_bP_x^T$, where P_x is a vector containing the x values of the control points and

$$M_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Multiplying the matrices gives

$$x(t) = (1-t)^3P_{x1} + 3t(t-1)^2P_{x2} + 3t^2(1-t)P_{x3} + t^3P_{x4}$$

There are similar equations for y and z.

In three dimensions, a surface patch is defined by 16 control points. (See fig. 3.) The four points along each edge of the patch describe Bézier curves; two points define the endpoints, and the other two determine the tangents of the curves at the endpoints. The equation for the bicubic Bézier patch is

$$x(s,t) = SM_bP_xM_b^TT^T$$

where P_x is a 4×4 matrix of the x-components of the control points. There are similar equations for $y(s,t)$ and $z(s,t)$.

The main advantages of using the Bézier polynomials as basis functions are that the shape of the patch can be easily changed by manipulating the control points and that the polyhedron formed by the control

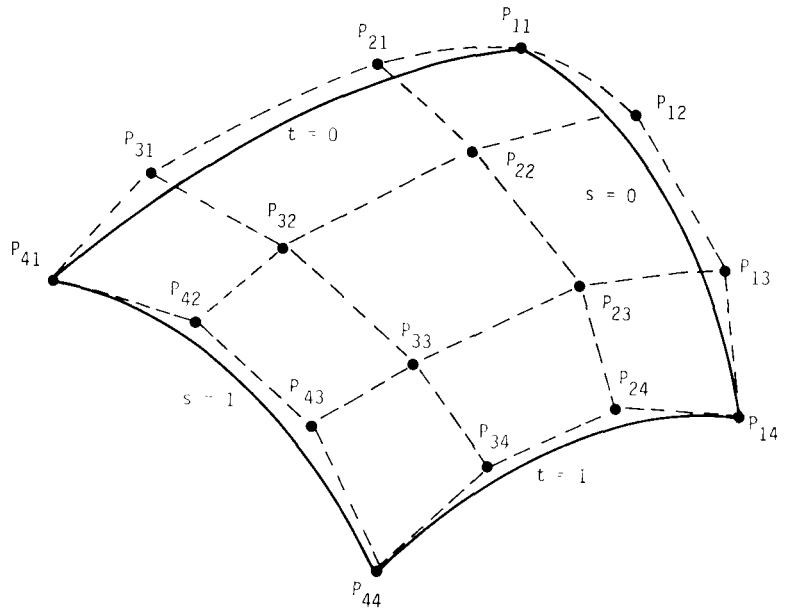


Figure 3. Bicubic Bézier patch with control points.

points is a convex hull which bounds the surface. The convex-hull property is useful in testing a patch for intersection with another patch or with a clipping plane. The convex hull is tested first, and only when it intersects is it necessary to test the actual patch.

Shape Generation

Since the same mathematical basis is used independently of the shape of the objects, the intersection algorithm can be developed and tested using simple shapes, such as cubes and spheres, and then applied to arbitrary shapes. The only requirement is that the objects be closed surfaces made up of adjoining patches.

As illustrated in figure 4, a cube is defined by six patches, one for each face of the cube. Since the faces are planar, the 16 control points for each face lie in the same plane. The points that define the sides of each patch may lie anywhere on the edges of the cube, with the endpoints on the corners, and the interior points of the patch may lie anywhere on the face. For convenience, they were assumed to be evenly spaced in this case.

A sphere consists of eight patches, each of which is constructed by rotating a 2-D Bézier curve approximation of a quarter-circle through an arc of 90°. Although an exact reproduction of a circle or a sphere is not possible with Bézier polynomials, the following technique produces an accurate approximation. The quarter-circle is defined by four Bézier points as shown in figure 5 for a unit circle. One of the defining parametric equations becomes

$$\begin{aligned} x(t) &= (1-t)^3 x_1 + 3t(t-1)^2 x_2 + 3t^2(1-t)x_3 + t^3 x_4 \\ &= 3t(t-1)^2\alpha + 3t^2(1-t) + t^3 \end{aligned}$$

At the midpoint of a quarter-circle, $t = 1/2$ and $x = \sqrt{2}/2$. Substituting these values into the above equation and solving yields

$$\alpha = \frac{4\sqrt{2}-4}{3} = 0.552285$$

For circles with a radius other than 1, the distance of the slope control points from the axes is α times the radius.

The result of rotating the circular arc through 90° to create a spherical segment is shown in figure 6. The control points for the edge curves $(P_{11}, P_{12}, P_{13}, P_{14};\ P_{14}, P_{24}, P_{34}, P_{44};\ P_{41}, P_{42}, P_{43}, P_{44})$ follow

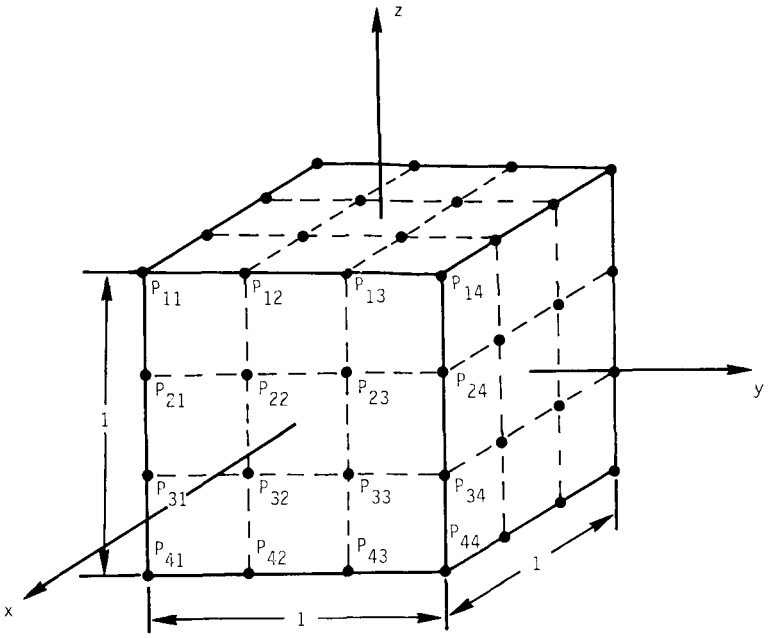


Figure 4. Patch representation of a cube.

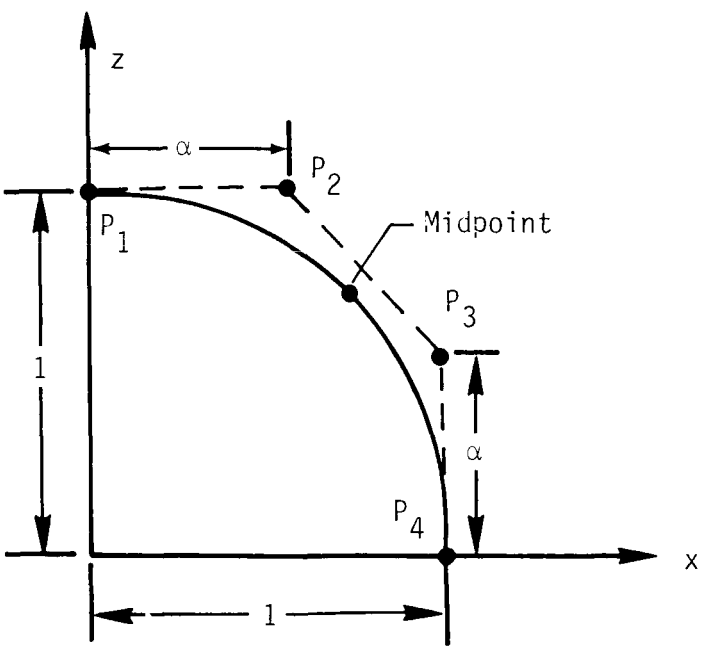


Figure 5. Bézier representation of a quarter-circle.

from the preceding paragraph, as they are 2-D circles. The locations P_{22}, P_{32}, P_{23}, and P_{33} are easily derived from the edge definitions by maintaining circular cross sections taken parallel to the x-y plane. For example, control points P_{22} and P_{32} are defined as if (P_{12}, P_{22}, P_{32}, P_{42}) corresponds to a circle of radius α; thus, the distance from P_{42} to P_{32} is α times the radius or α^2. The entire patch is rotated about the axes to generate the seven additional patches required to describe a complete sphere. Care must be taken in the ordering of the control points. The points in figure 6 are ordered in a way that ensures that the calculated surface normal always points to the outside of the closed surface of revolution.

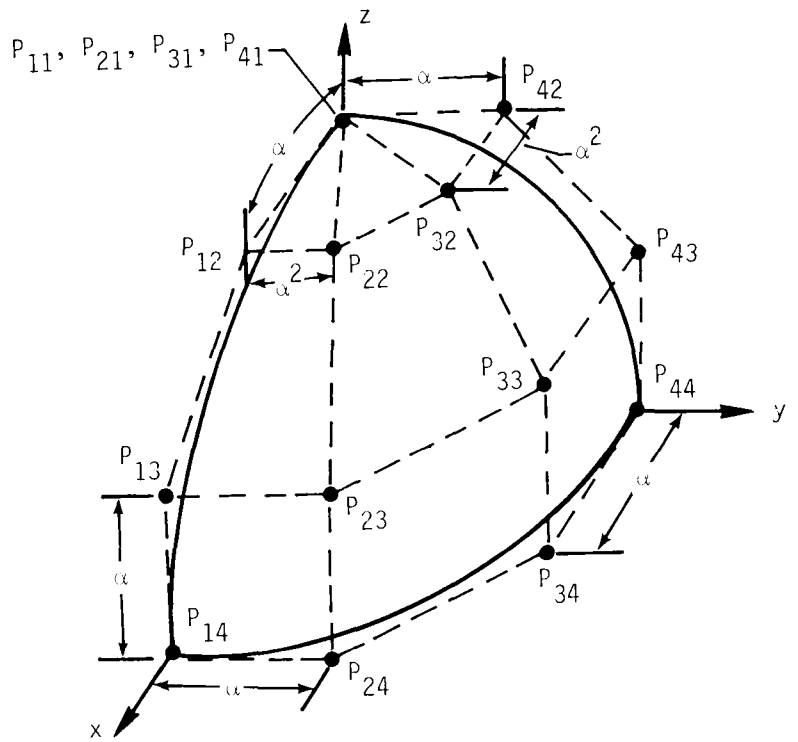


Figure 6. Bézier representation of a 1/8 sphere.

The same procedure can be used to generate a body of revolution from any 2-D Bézier curve. Figure 7 shows a bell-shaped object generated from a Bézier curve defined in the x-z plane and rotated through 90°. As with the sphere, the surface segment can be rotated about the axes to complete the entire body of revolution. The endpoints of the original planar curve are placed on the axes to ensure that a closed surface is formed.

Object Manipulation and Drawing

Computer programs were written for the interactive generation, manipulation, and viewing of the objects used in the intersection determination. The programs were written in FORTRAN 77 on a Prime 850 minicomputer. With these programs and a graphics terminal, the user can construct two 3-D objects, each of which can be either a cube or a body of revolution. For describing a body of revolution, the program places the y and z axes on the screen and, using a cross-hair cursor, the user inputs the four Bézier control points of the originating curve. These points are then individually moved, and the curve is redrawn until the desired curve is produced. A unit cube as described previously and shown in figure 4 is provided by the program when selected by the user.

Once two objects are created, they can be individually moved, scaled, or rotated by the user until they are in the desired position and orientation with respect to one another. The objects are displayed on the terminal screen by drawing the patches as a series of curves defined by constant s and constant t (ref. 4). The result is a grid pattern on the surface of each object (fig. 8) for a cube and a body of revolution.

Object Intersection

General Algorithm

When portions of two 3-D objects occupy the same space, a common line of intersection is created on the surfaces, and some of the closed surface of each object is contained within the other object. An algorithm is used to determine the line of intersection (ref. 2). Each patch of one object is compared with every patch of the other object, and the intersection between the two patches is calculated if it exists.

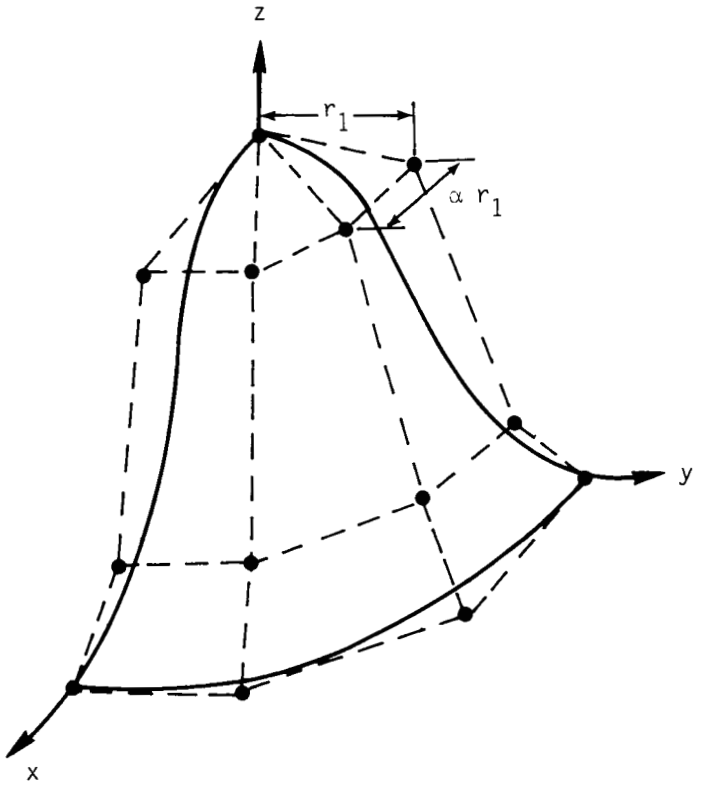


Figure 7. Body of revolution generated from an arbitrary Bézier curve.

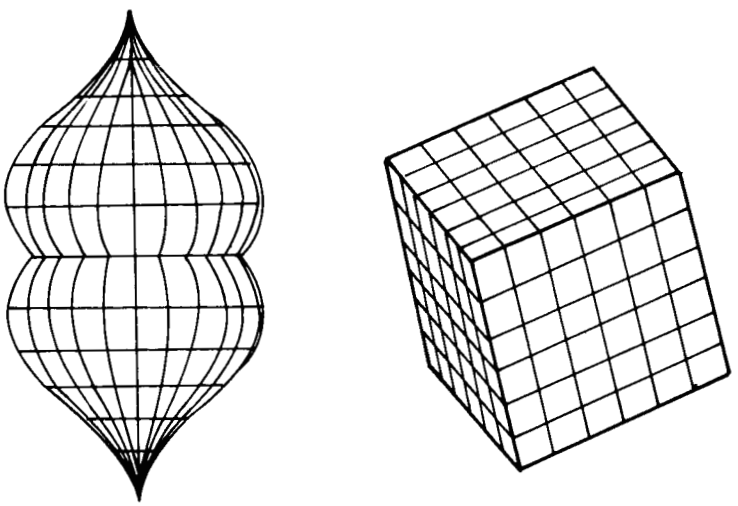


Figure 8. Sample display of Bézier surfaces.

The complete curve of intersection consists of the line segments determined from all the patch-to-patch intersections.

There is no known way to analytically determine the curve of intersection between two bicubic patches. Therefore, the patches are broken down, or subdivided, into smaller patches that are planar within some tolerance. This subdivision results in the simpler problem of calculating the straight line of intersection between planar polygons. Once the intersection is found, the orientation is determined; that is, it is determined which portion of the patch is inside the other object. Likewise, if no intersection is found, the

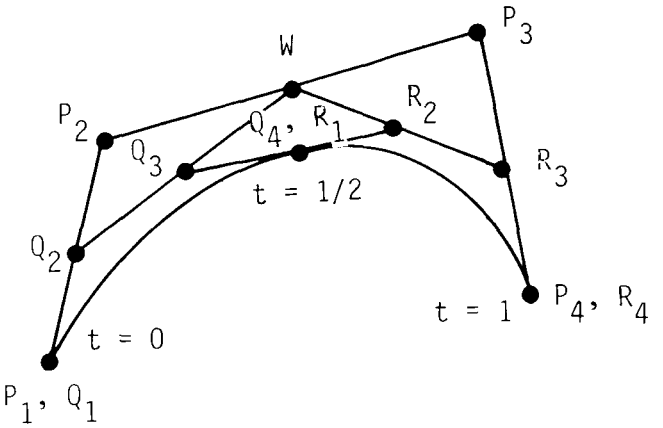


Figure 9. Subdivision of a Bézier curve.

orientation of the entire patch with respect to the other object is calculated. The details of each part of the intersection algorithm, including the handling of special cases, are presented in the following section. Each of the parts is described in words and by an algorithmic pseudo-code description.

Patch Subdivision

The subdivision process consists of dividing the patches of both objects until they are approximately planar or until it is determined that they cannot possibly be involved in the intersection. A patch that can be eliminated from the intersection at this stage is termed separable. A fast method of subdividing bicubic patches (ref. 3) is applied to the problem. An example of using this algorithm to subdivide a 2-D Bézier curve is shown in figure 9. The original curve defined by the control points P_i is divided at the point where the value of the parametric variable t is 1/2. The control points of the two smaller curves, Q_i for $t = 0$ to $t = 1/2$ and R_i for $t = 1/2$ to $t = 1$, are easily computed as follows:

$$
\begin{aligned}
Q_1 &= P_1 \\
Q_2 &= (P_1 + P_2)/2 \\
Q_3 &= (Q_2 + W)/2 \qquad \text{(where } W = (P_2 + P_3)/2\text{)} \\
R_4 &= P_4 \\
R_3 &= (P_3 + P_4)/2 \\
R_2 &= (R_3 + W)/2 \\
R_1 &= Q_4 = (Q_3 + R_2)/2
\end{aligned}
$$

The method can be extended to surfaces in three dimensions where the subdivision of the boundary curves is similar to the 2-D case. As a result, the bicubic patch is divided into four adjoining patches as shown in figure 10. These four patches together reproduce the original single patch exactly, and can replace it in the analysis. Each of the new patches can now be tested for planarity or separability and can be further subdivided if necessary. If a patch is determined to be planar or separable, this information is placed in the data structure, the details of which are discussed subsequently. The subdivision proceeds recursively until the fate of all the patches in both objects is determined, at which time the objects consist of a mixture of bicubic patches and planar polygons in the neighborhood of the line of intersection. The subdivision can be described in algorithmic form as follows:

```
Procedure subdivision
  Begin
    For each object
      For each patch in the object
```

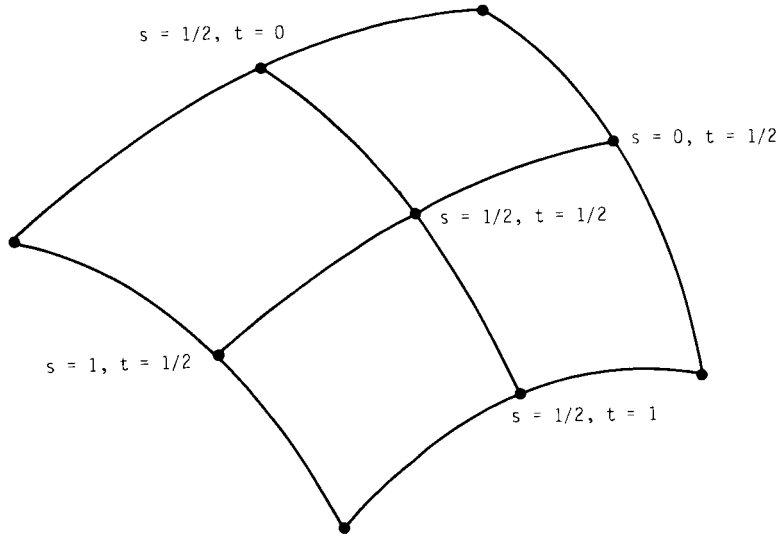


Figure 10. Subdivision of a Bézier patch.

```
Procedure sdtest
  Begin
    Is the patch separable from the other object?
      Case yes:
        Set orientation to outside
        Enter patch into data structure
        Return
      Case no:
        Is the patch nearly planar?
          Case yes:
            Set intersection involvement flag
            Enter patch into data structure
            Return
          Case no:
            Subdivide patch
            For each subpatch
              Recursively call sdtest
End sdtest
End subdivision
```

At this point, the separability determination is very simple. The control points of the patch in question are compared with the bounding box formed from the maximum and minimum x, y, and z values of the other object. If all the control points are larger than the maximum value or smaller than the minimum value of any coordinate, the patch is completely outside the other object, and no intersection is possible. It can then be flagged as not being in the intersection and as having an outside orientation.

The planarity test is more complicated. The patch is first assumed to be planar. The equation of the plane is $Ax + By + Cz + D = 0$, where the first three coefficients are determined from the cross product of the two diagonal vectors connecting the opposite corner points of the patch and where D is calculated by substituting one of the corner points into the equation. The distance of each control point from the plane is then calculated, and the distance of the point farthest from the plane on one side is added to the distance of the point farthest from the plane on the other side. This addition yields a value that can be thought of as the "thickness" of the patch. If the thickness is less than the desired tolerance, the patch is assumed to be a planar quadrilateral for the remainder of the analysis. The patch is flagged as being

a possible contributor to the intersection, and the points defining the patch are entered into the data structure.

Data Structure

During the subdivision process, the patches are divided into four subpatches; therefore, a quadtree was chosen as the data structure which would best represent the data logically (ref. 2). A quadtree is a data tree in which each node has four children. When a patch is determined to be separable or planar and is no longer subdivided, the data for that patch are put into a leaf node of the tree. The structure of the data record is discussed subsequently.

Figure 11 illustrates how the quadtree is recursively built up during the subdivision of one patch. The original patch is shown with a curve of intersection with another patch. Although the line of intersection is generally not known a priori, it is assumed to be known for the purpose of the illustration. As the subdivision proceeds, the tree grows down and to the left until a patch does not require subdivision. A leaf node is generated there, and the process goes back up the tree and restarts at the next available branch. In the figure, the patches and their corresponding leaf nodes are numbered in the order they are generated. The leaf nodes designated by the open boxes represent patches that are separable and not involved in the intersection, whereas the closed boxes represent leaf nodes that are planar and possibly involved in the intersection.

The structure of the records at each leaf node that is involved in the intersection follows the form of the data generated during the intersection calculation. When a planar patch from one object intersects a patch from the other object, a line of intersection is formed in one of three ways. (See fig. 12.) Two intersection points which define the line segment are found and placed into a data list. More than two points can be found if two edges intersect each other, and this is handled as a special case. The two points which define an intersection line segment are referred to as partners. The intersection points produced by the edges of a patch are referred to as exterior intersection points for that patch. Points within a patch generated by the edges of other patches are interior intersection points. Intersection records for the two patches are constructed which refer to the intersection points. Since any given patch may intersect with more than one patch from the other object, several of these intersection records are produced and then connected by a doubly linked list structure.

In the FORTRAN program, the patch information is kept in a 3-D array, rnodeleaf(i,j,k), with structure as follows:

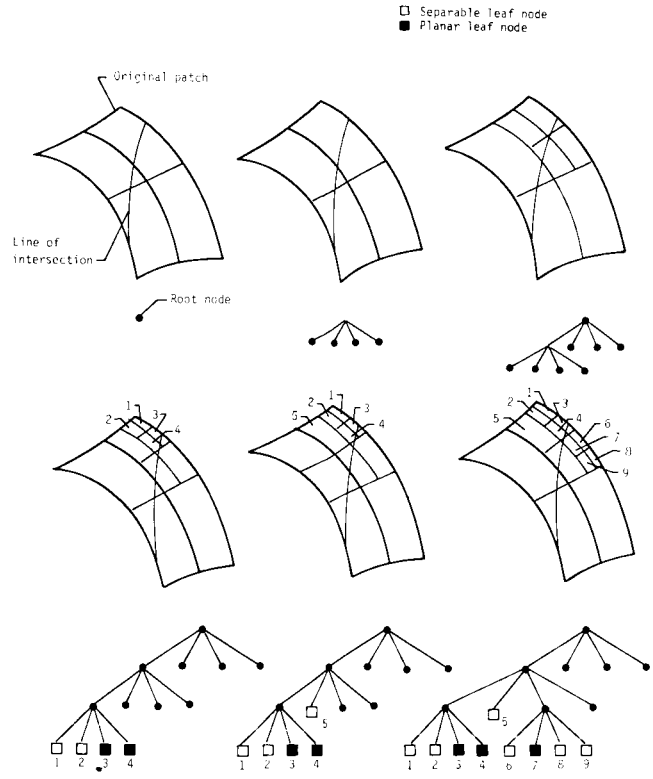
First dimension(i):
- 1...48 - Coordinates of 16 control points of patch $(x_1, y_1, z_1, \ldots, x_{16}, y_{16}, z_{16})$
- 49 - Orientation indicator of patch (1 = "Outside", −1 = "Inside")
- 50 - Pointer to first intersection record for this patch (if null, patch is not involved in intersection)

Second dimension(j):
- Patch (node) number

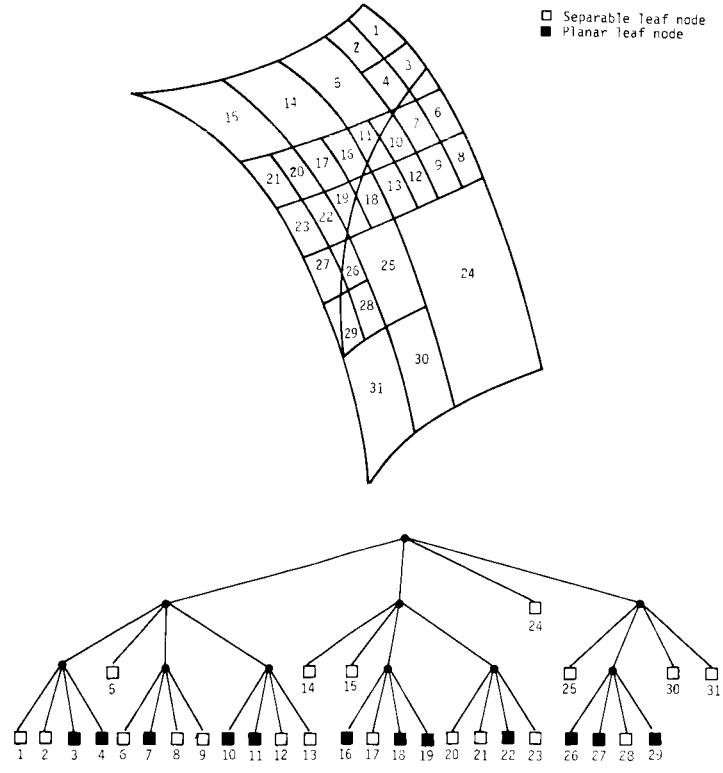
Third dimension(k):
- Object number

Leaf nodes involved in the intersection have information kept in an intersection record in the form of a companion $9 \times 1 \times 1$ array, intsnode(i,j,k), with structure as follows:

First dimension(i):
- 1 Pointer to previous intersection record for specific patch (null if this is the first)
- 2 Pointer to an intersection point
- 3 Pointer to an intersection point
- 4 Integer value (0...3) dependent on point intersection involvement
  - 1 Exterior intersection point given by the pointer in index 2. (See fig. 12(b).)
  - 2 Exterior intersection point given by the pointer in index 3. (See fig. 12(b).)
  - 3 Exterior intersection points given by pointers in indices 2 and 3. (See fig. 12(a).)
  - 0 Interior intersection points given by pointers in indices 2 and 3. (See fig. 12(c).)

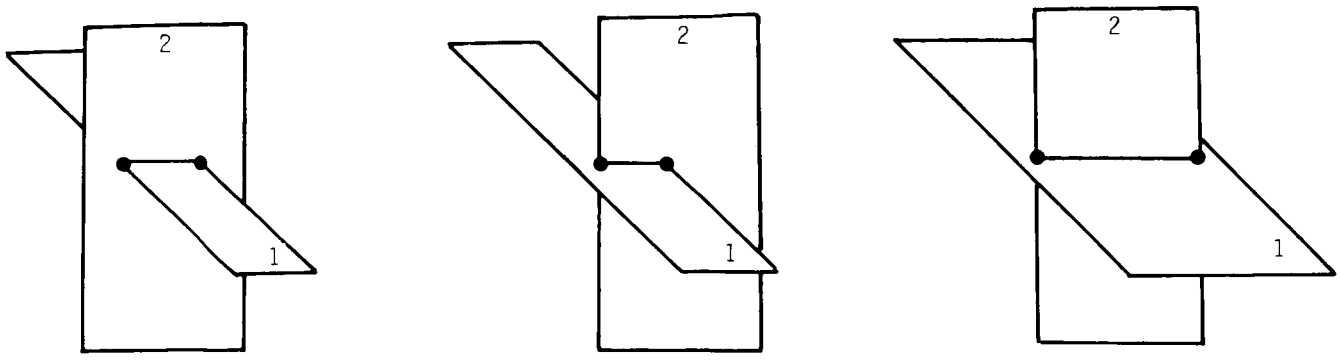


(a) Recursive building of quadtree during subdivision.



(b) Completed quadtree.

Figure 11. Example of quadtree data structure.



(a) Both endpoints from patch1; two exterior points for patch1.

(b) One endpoint from each patch; one exterior and one interior point for each patch.

(c) Both endpoints from patch2; two interior points for patch1.

Figure 12. Planar intersection possibilities.

5 –Patch edge number corresponding to point located through index 2 (0 if index 4 is 0 or 2)
6 –Patch edge number corresponding to point located through index 3 (0 if index 4 is 0 or 1)
7 –Orientation corresponding to point located through index 2
8 –Orientation corresponding to point located through index 3
9 Pointer to next intersection record for specific patch (null if this is the last)

Second dimension(j):
Intersection record pointer

Third dimension(k):
Object number

Figure 13 displays the manner in which the preceding data structures interact. Since leaf node 1 is designated separable, the only information needed about this patch is the 16 control-point values and the orientation of the patch with respect to the other object. No intersection information exists; thus, that pointer is null. Leaf node 3, however, is not separable. The 16 control-point values of this patch must also be kept, but the orientation of the patch cannot be determined at that time. A pointer to the intersection data structure is set upon intersection calculation. This pointer refers to the first block of intersection information about the patch. If more than one block of information exists, as in figure 13, the data structures are linked together through forward and backward pointers. For example, intersection record 2 is the first of two blocks of intersection information for the leaf node. Therefore, the backward pointer is set to null, and the forward pointer is set to intersection record 3. Intersection record 3 in turn has a backward pointer set to intersection record 2 and a forward pointer set to null, since no other intersection information about the leaf node exists. Pointers to intersection points involved with the patch are also kept in the intersection data structure. These pointers are directed to a two-dimensional array, pointints(i,j), that contains the coordinates of the intersection points. As shown in figure 13, the pointers correspond to the column index in the array. For example, intersection point 3 has x coordinate in pointints(1,3), y coordinate in pointints(2,3), and z coordinate in pointints(3,3).

Intersection Algorithm

The general intersection algorithm described previously is presented in more detail below and is followed by the intersection possibilities algorithm. The intersection possibilities procedure determines the intersection, if any, between two given planar patches. A series of tests are performed to immediately eliminate nodes from the comparison process if an intersection cannot exist. To reduce the amount of computation, the tests are performed in order of increasing complexity. Explanations of these tests follow the intersection algorithms. The intersection algorithms are as follows:

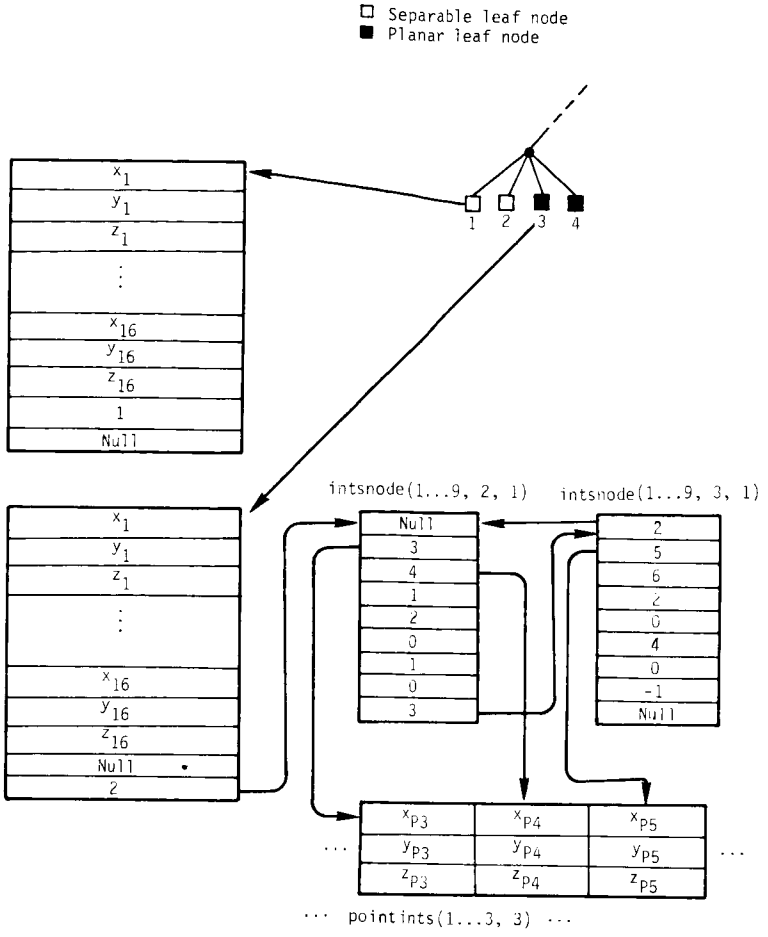


Figure 13. Data structure interaction.

Procedure int_test
  Begin 1
    For each leaf node of object one
      Begin 2
        Was this object one node declared separable by the subdivision testing?
          Case yes:
            Continue with next leaf node
          Case no:
            For each leaf node of object two
              Begin 3
                Was this object two node declared separable by the subdivision testing?
                  Case yes:
                    Continue with next leaf node
                  Case no:
                    Evaluate intersection possibilities between the two nodes (procedure i_test)
              End 3
        Is this object one node involved in the intersection?
          Case yes:
            Continue with next leaf node
          Case no:
            Reset orientation indicator (evaluate mid-distance test, if orientation indicator is less than 0,
                Indicator = −1, otherwise Indicator = 1)
      End 2

```
    For each leaf node of object two
      Begin 4
        Is this object two node involved in the intersection?
          Case yes:
            Continue with next leaf node
          Case no:
            Set orientation indicator (evaluate mid-distance test)
      End 4
  End 1

Procedure i_test(patch1,patch2)
  Begin 1
    Calculate orientation of both patch1 and patch2 in case no intersection points between the two patches
        are found
    Are the bounding boxes separable (box test)?
      Case yes:
        Return (no intersection)
      Case no:
        Are the patches parallel to one another (parallel plane test)?
          Case yes:
            Return (no intersection)
          Case no:
            Does patch2 cross the bounding planes of patch1?
              Case no:
                Return (no intersection)
              Case yes:
                Evaluate the patch edge test for patch1 (procedure edge_test)
                Evaluate the patch edge test for patch2 (procedure edge_test)
                Were any intersection points found by either patch edge test?
                  Case no:
                    Return (no intersection)
                  Case yes:
                    Was only one intersection point found?
                      Case yes:
                        Eliminate the point from consideration as if no intersection had occurred
                        Return
                      Case no:
                        Are the two intersection points the same point (i.e., does an edge from patch1
                            intersect an edge from patch2)?
                          Case yes:
                            Eliminate both points from consideration as if no intersection had occurred
                            Return
                          Case no:
                            Continue
  End 1
```

The box test represents patch1 and patch2 as 3-D bounding boxes (minimum and maximum coordinates) in the same manner as during the subdivision. If the boxes intersect, the patches are not separable.

The orientation of a patch is calculated by the middistance test. An important factor in determining the final orientation of patch1 is to find the patch2 that lies closest to it. (See fig. 14.) This is done by calculating the distance between the midpoints of patch1 and each of the patches in the second object

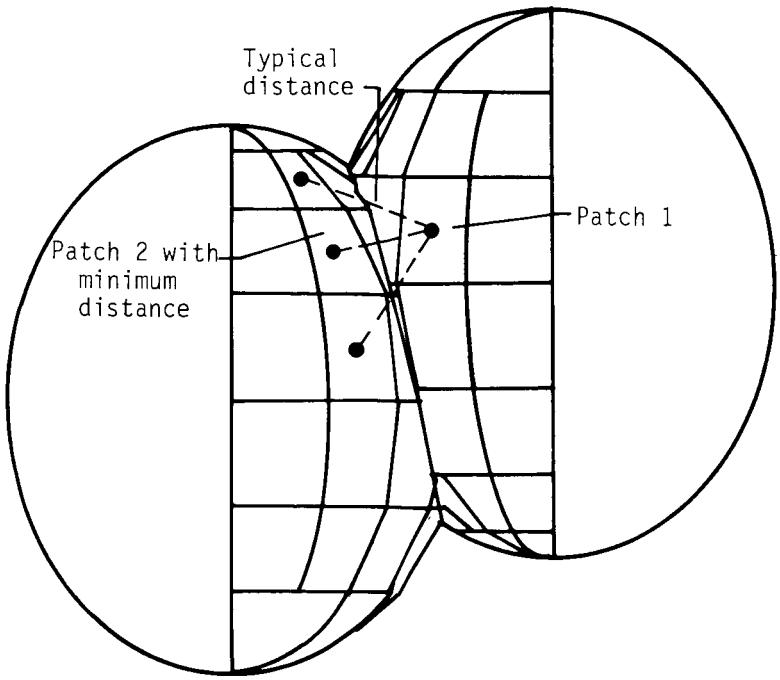


Figure 14. Orientation of a patch not separable and not in intersection.

and by updating the orientation indicator as the distance decreases. The midpoint is computed as follows:

$$x = (x_{\text{control\_pt1}} + x_{\text{control\_pt4}} + x_{\text{control\_pt16}} + x_{\text{control\_pt13}})/4$$
$$y = (y_{\text{control\_pt1}} + y_{\text{control\_pt4}} + y_{\text{control\_pt16}} + y_{\text{control\_pt13}})/4$$
$$z = (z_{\text{control\_pt1}} + z_{\text{control\_pt4}} + z_{\text{control\_pt16}} + z_{\text{control\_pt13}})/4$$

The distance is then calculated by

$$d = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

where (x_1, y_1, z_1) and (x_2, y_2, z_2) are the midpoints of patch1 and patch2, respectively. If d is greater than the absolute value of the previously determined orientation distance, no update of the orientation indicator is needed. Otherwise, the midpoint of patch1 is substituted into the planar equation for patch2 to determine on which side of patch2 the point is located. The equation then becomes $E = Ax_1 + By_1 + Cz_1 - D$, where A, B, C, and D are the planar coefficients of patch2. Patch1 is inside the other object if E is negative. It is outside the object if E is positive. Finally, the orientation indicator is updated by multiplying d by the sign of E. For example, in figure 14, the distances between patch1 and several patches in object2 are indicated by dashed lines. In this case, patch1 is determined to be outside of object2.

The parallel-plane test is a simple test to determine if patch1 and patch2 are parallel. If the planar coefficients of the two patches are multiples of one another within a tolerance, the two patches are parallel and are defined as separable.

The bounding-plane test is a separability test discussed by Carlson (ref. 2). It is similar to the patch thickness calculation that was used in the subdivision process in that the control points farthest away from each side of the planar approximation of the patch are used. Two planes are created, one passing through each of these two control points parallel to the planar approximation of patch1. (See fig. 15.) If any of the control points from patch2 lie between the two planes, the patches are not separable.

The patch edge test determines which edges, if any, of patch1 intersect patch2. The edges are defined as the following lines:

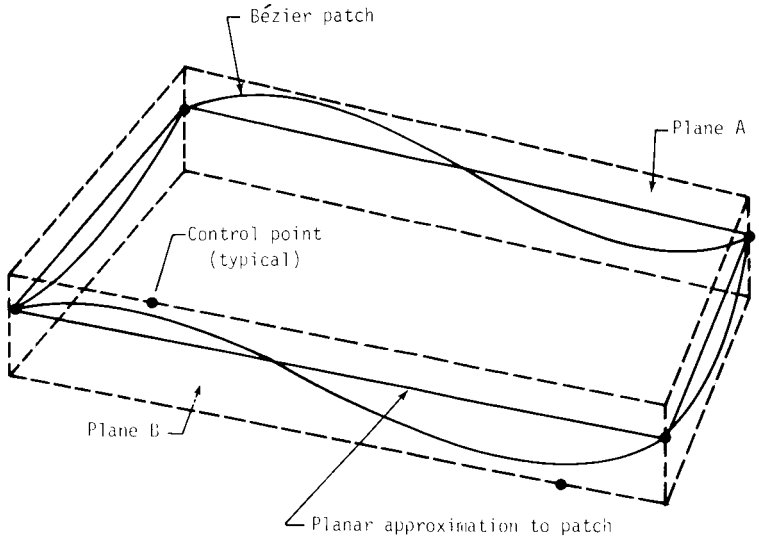


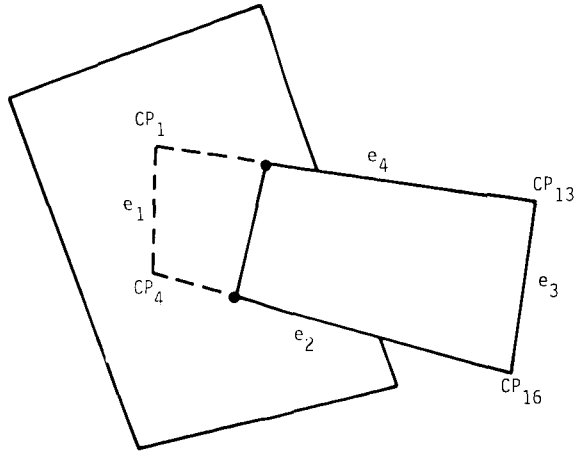
Figure 15. Bounding-plane separability test.

e_1: control_pt1 to control_pt4
e_2: control_pt4 to control_pt16
e_3: control_pt16 to control_pt13
e_4: control_pt13 to control_pt1

The endpoints of the edges are substituted into the planar equation for the other patch. If the results have different signs, the edge intersects the plane. The signs of the results also give the orientation of the edge, that is, whether it is going into or coming out of the other object. If the result is negative from the first end point and positive from the second, the edge is coming out of the other object. If the result is positive from the first end point and negative from the second, the edge is going into the other object. (See fig. 16.) The test is described in algorithmic form as follows:

Procedure edge_test (patch1,patch2)
Begin 1
For each edge of patch1
Begin 2
Evaluate planar equation of patch2 at the endpoints of the edge ($E_1 = Ax_1 + By_1 + Cz_1 - D$, $E_2 = Ax_2 + By_2 + Cz_2 - D$, where (x_1, y_1, z_1) and (x_2, y_2, z_2) are the edge endpoints and A, B, C, and D are the planar coefficients of patch2)
Does either endpoint lie on patch2 (is E_1 or $E_2 = 0$)?
Case yes:
No intersection (if an edge endpoint lies on patch2, it cannot pass through patch2)
Case no:
Does the entire edge lie on one side of the plane (is the sign of E_1 equal to the sign of E_2)?
Case yes:
No intersection
Case no:
Evaluate intersection-point calculation and containment (procedure int_point), the sign of E_1 is used as an orientation indicator if an intersection point is determined
End 2
End 1

The final procedure is to calculate and evaluate the intersection point between an edge from patch1 and the planar representation of patch2. The computed intersection point is kept only if it lies within the polygonal boundaries of patch2. However, only two distinct intersection points can exist between two planar patches. Figure 17 illustrates several features of the intersection-point calculation; P_{11} is a patch



Edge	Sign of E_1	Sign of E_2	Orientation	
1	-	-		
2	-	+	-	Coming out
3	+	+		
4	+	-	+	Going in

Figure 16. Patch edge test.

from object one, and P_{21} and P_{22} are from object two. When P_{11} is compared with P_{21}, the two points defining the intersection line are found. The intersection of the third edge (e_3) of P_{11} with P_{21} produces I_1, and the intersection of the third edge of P_{21} with P_{11} produces I_2. These two points are partners and are saved in the intersection-point list. As shown by comparing P_{11} with P_{22}, the first edge of P_{22} generates I_3 when it passes through P_{11}. The first edge of P_{11} and the third edge of P_{22} intersect and produce two points. Since they are duplicates, one is discarded, and one is labeled I_4 and put into the list as a partner of I_3. Since P_{21} and P_{22} are adjacent, I_2 and I_3 coincide. This information is used in a point-ordering scheme that connects the individual line segments to form a continuous intersection curve.

The procedure is more complicated than described. More potential intersection points are calculated than are shown in figure 17. For instance, the first edge of P_{21} intersects the plane that contains P_{11}. This point of intersection is calculated, but is discarded because it falls outside the boundaries of P_{11}. Many points are calculated and discarded for this reason. The subsequent algorithms in this section explain the procedure used in more detail. The first of these algorithms is as follows:

```
Procedure int_point(edge_no,patch1,orien1,patch2)
  Begin 1
    Calculate the intersection point of the edge from patch1 and the planar representation of patch2
    Project patch2 onto the coordinate plane which produces the largest projected area
    Does (u_p, v_p), the projected intersection point, lie within the projected region given by procedure re-
        gion(uarray,varray)?
      Case no:
        Return (no intersection)
      Case yes:
        Is this the first intersection point found between the two patches?
          Case yes:
            Set pointers and other information into intersection node
          Case no:
```

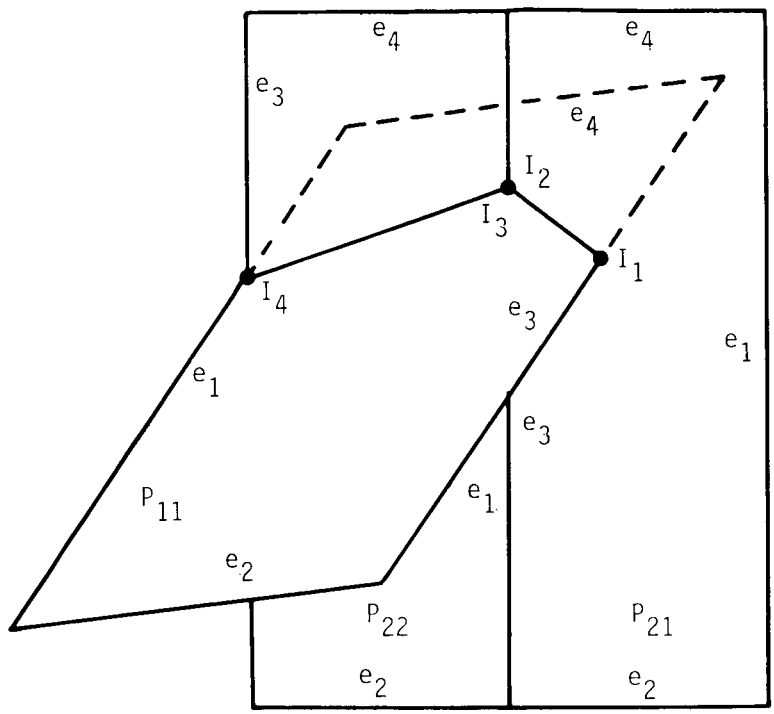


Figure 17. Intersection points and polygon reconstruction.

```
        Is this the second intersection point found?
          Case yes:
            Input information into intersection node
          Case no:
            Determine which two of the three intersection points are identical
            Input information into intersection node accordingly
      Return(intersection)
End 1
```

The procedure region used to determine whether the intersection point is within the 2-D planar polygon is well-known. As shown in figure 18, for each edge in the bounding polygon, a triangle is constructed from the endpoints of the edge and the intersection point. The angles opposite the edges, indicated by θ_i in the figure, are calculated and added; if the result is 360°, the point is within the polygon. It is possible that, because of round-off error in the computer, a point inside the polygon but close to the boundary may be determined to be outside. Thus, any point within a tolerance of the boundary is considered to be inside. The second algorithm used for explaining the intersection-point calculation is as follows:

```
Procedure region (up,vp,uarray,varray)
  Begin 1
    Initialize difference-angle sum variable (DSUM) to 0
    Calculate a direction vector from the intersection point to each of the four endpoints of the region
    For each direction vector (dv)
      Begin 2
        Calculate the difference angle (dangle) from dvi to its adjacent direction vector (dvi+1)
        Is the abs(dangle) = 180 within the tolerance?
          Case yes:
            Return (intersection point lies on an edge of the region)
          Case no:
            Increment dsum by dangle
```

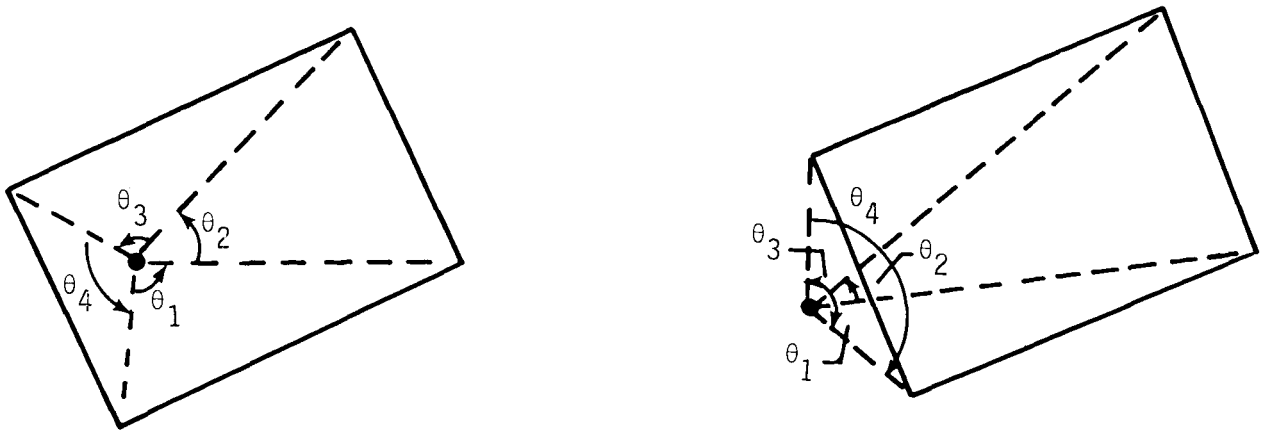


Figure 18. Testing for a point inside a polygon.

Continue with next direction vector
End 2
Is the sum of the difference angles equal to 360 within the tolerance?
Case yes:
Return (intersection point lies within the region)
Case no:
Does the intersection point lie outside the region but within tolerance of an edge, and is it therefore considered within the region?
Case yes:
Return (intersection point lies within the region)
Case no:
Return (intersection point lies outside the region)
End 1

Reconstruction of Polygons Involved in the Intersection

When the intersection processing is complete, the patches that were involved have a number of intersection points associated with them. In general, each patch has the two points at which its edges intersect the other object. Each patch may also have points interior to the patch. These points form interior intersection segments and are generated by patches from the other object. Several special cases may infrequently occur, particularly if a patch is large compared with the patches from the other object. For example, holes or islands may occur if no edges of a patch intersect the other object but if edges from the other object intersect it.

The first step in the construction of polygons formed by the intersection of two objects is the proper ordering of patch vertices and intersection points to define the patch regions which make up the final object. Each of the original four-sided patches becomes at least two n-sided polygons, with at least one polygon oriented outside the other object and one inside.

The technique for ordering the points consists of several steps. First, the intersection points on the edges of a patch (exterior intersection points) are put into a list ordered by edge number. If more than one exterior intersection point lies on an edge, those points are ordered according to increasing distance from the first endpoint of the edge. The first point on the list is the first vertex of the new polygon. Depending on the orientation of this point, either the edges of the patch or the interior intersection line segments are followed until another exterior intersection point is encountered. The recursive procedure winseg orders the interior intersection points and creates an intersection curve in the patch. Intersection segments are matched according to endpoints, and the procedure stops when an exterior intersection point for the patch is located. The process is then repeated until all edges and exterior intersection points have been used.

If the edges are followed, the endpoints of the edges are added to the vertices of the new polygon. If the intersection line segments are followed, the interior intersection points become vertices.

This process can be illustrated by using the patches in figure 17. If it is assumed that polygons are being constructed from P_{11}, then the ordered list of exterior intersection points for P_{11} consists of I_4 followed by I_1. The starting vertex for the new polygon is I_4, and its orientation is negative, or coming out (see procedure edge_test). If the polygon that is outside the other object is to be found, the edges of the original P_{11} are followed, adding each vertex to the new polygon, until the next intersection point, I_1, is encountered. Since the orientation of I_1 is positive, or going in, its partner, I_2, is designated as the next vertex. Since I_2 is not an exterior intersection point, the list of interior points is searched until a match is found, in this case I_3. The partner of I_3 is an exterior point, I_4, so the process terminates with a complete five-sided polygon. If the inside polygon is to be reconstructed, the interior intersection lines are followed from I_4 to I_1, and then the edges of the original patch are followed. A special case in this method of point ordering involves the formation of islands or holes within a polygon. If only interior intersection points exist for a patch, then an island must be within that patch. If the area outside the island is to be saved, point ordering begins with the four patch vertices and ends with the ordering of interior intersection points in a counterclockwise direction. However, if the island area is to be saved, point ordering includes only the interior intersection points, but they are ordered in a clockwise direction.

The ordering procedure is presented in detail in the following three algorithms:

```
Procedure ordrpy(patch1,leaf1,orien1)
For each edge of patch1, i = 1 to 4 do
  Begin 1
    For each exterior intersection point of patch1
      Begin 2
        Does this intersection point come from edge_i?
          Case no:
            Continue with next intersection point
          Case yes:
            j = j + 1
            Save index(j), the pointer to the intersection point, in a list
            Visited_list(j) = False
      End 2
  End 1
Total_ext = j
Is there an odd number of exterior intersection points?
  Case yes:
    Return(error)
  Case no:
    Continue
Order exterior intersection points which lie on the same edge
Save a list of the interior intersection segments of patch1
Do any interior intersection points exist?
  Case no:
    nzero = 0
  Case yes:
    nzero = 41
    For each interior intersection segment
      Begin 3
        nzero = nzero - 1
        Save the pointer information, ndex(nzero)
        Visited_list(nzero) = False
      End 3
(Create polygons dependent upon the declared orientation, (orien1) with ordered vertices in a list orderp.)
jj = 1
```

j = 1
Until j > total_ext do
(Do until each exterior intersection point(EIP) has been visited, i.e., used to form a polygon)
  Begin 4
    Has EIP_j been visited?
      Case yes:
        Continue with next point
      Case no:
        If this is the first point of the polygon (i.e., jj = 1), save a pointer ($edge_{beg}$) to the edge number of this point
        Is this point's orientation the same as orien1?
          Case no:
            $orderp_{jj} = EIP_j$
            jj = jj + 1
            Add EIP_j to the list of intersection points visited (Visited_list(j) = True)
          Case yes:
            $orderp_{jj} = vertex\_pt_{edge\#}$, where edge# is the edge number EIP_j came from
            $orderp_{jj+1} = EIP_j$
            $orderp_{jj+2}$ = Partner of EIP_j
            Visited_list (j) = True
            Is the partner also an external intersection point?
              Case no:
                Save pointers to EIP_j and its partner (m = ndex(j), m2 = ndex(j)+1)
                jj = jj + 2
                Continue creating the polygon by ordering exterior and interior intersection points (procedure winseg)
                Update j to point to the exterior intersection point procedure winseg returned from its ordering (i.e., for what j does ndex(j) = m?)
              Case yes:
                Update j to point to the partner of EIP_j
                (For what j does ndex(j) = m2?)
            Has EIP_j been visited?
              Case yes:
                (Have completed an ordered polygon)
                Save this polygon
                Save the planar coefficients of patch1
                Initialize to start a new polygon
                j = 1
                Return to "Begin 4"
              Case no:
                Visited_list(j) = True
        Save a pointer ($edge_{last}$) to the edge number of this point
        For n = j + 1 to total_ext do
          Begin 5
        Has EIP_n been visited?
              Case yes:
                Continue with next point in list
              Case no:
                Are $edge_n$ and $edge_{last}$ the same?
                  Case no:
                    Continue ordering polygon points by including into orderp the vertex points from $edge_{last}$ + 1 through $edge_n$ − 1
                  Case yes:
                    Continue
                j = n

Return to "Begin 4"
End 5
Enclose the polygon by including in orderp the vertex points from $edge_{last} + 1$ through $edge_{beg}$
Save this polygon
Save the planar coefficients of patch1
Initialize to start a new polygon
Continue with j = 1
End 4
Is total_ext > 0?
Case yes:
Have all intersection points (exterior and interior) been visited?
Case yes:
Return
Case no:
An island exists within the polygon just ordered, point ordering continues
Case no:
An island exists within the original patch1, point ordering begins (procedure island)
Return

Procedure winseg (jj,orderp,leaf1,ndex,nzero,visited_list,m,m2)
l = leaf1
For each intersection node leaf involved with patch1
Begin 1
Case 1:
$Orderp_{jj}$ = Neither endpoint of the intersection segment in intsnode(...,l,...)
Continue with next intersection node leaf (l =intsnode(9,l,...))
Case 2:
$orderp_{jj}$ = First endpoint of the intersection segment in intsnode(...,l,...) and $orderp_{jj}$ = Second endpoint, or $orderp_{jj}$ = First endpoint and the second endpoint but the first endpoint is an exterior intersection point and the second endpoint is an interior intersection point
Does $orderp_{jj-1}$ = First endpoint?
Case yes:
Continue with next intersection node leaf (l = intsnode(9,l,...)) (match is with the same segment just inputted into orderp)
Case no:
jj = jj + 1
$orderp_{jj}$ = First endpoint
Is this point an exterior intersection point?
Case yes:
Return(match made)
Case no:
Is the second endpoint an interior intersection point?
Case no:
Continue
Case yes:
Add this interior intersection segment to the visited list
Call winseg
Return
Case 3:
$orderp_{jj}$ = First endpoint of the intersection segment in intsnode(...,l,...) and $orderp_{jj}$ = Second endpoint
Or $orderp_{jj}$ = First endpoint and the second endpoint and either both endpoints are exterior intersection points or both are interior intersection points or only the second endpoint is an exterior intersection point
Does $orderp_{jj-1}$ = Second endpoint?

```
        Case yes:
          (Match is with the same segment just inputted into orderp)
          Continue with next intersection node leaf
        Case no:
          jj = jj + 1
          orderp_jj = Second endpoint
          Is this point an exterior intersection point?
            Case yes:
              Return(match made)
            Case no:
              Is the first endpoint an interior intersection point?
                Case no:
                  Continue
                Case yes:
                  Add this interior intersection segment to the visited list
              Call winseg
              Return
  End 1
  No match within tolerance for orderp_jj due to gapping
  Return(error)

Procedure island (jj,orderp,leaf1,ndex,nzero,visited_list,m,m2)
  Begin 1
    Is the island a hole cut out of patch1 (orien1=1)?
      Case yes:
        (The patch area outside the island is to be saved)
        orderp_1 = vertex_pt_1
        orderp_2 = vertex_pt_2
        orderp_3 = vertex_pt_3
        orderp_4 = vertex_pt_4
        orderp_5 = vertex_pt_1
        orderp_6 = First interior intersection point
        Add this point to the visited list
        orderp_7 = Interior intersection point's partner
        Save the planar coefficients for patch1
        Continue creating the polygon by ordering the island points in counterclockwise direction
      Case no:
        (The island itself is to be saved, orien1 = -1)
        orderp_1 = The first interior intersection point for this specific patch
        Add this point to the visited list
        orderp_2 = The interior intersection point's partner
        Save the planar coefficients for patch1
        Continue creating the polygon by ordering the island points in clockwise direction
    Input the last point of the polygon 2 more times
    Return
  End 1
```

Because of gaps between patches, caused by nonuniform patch subdivision, a matching intersection point cannot always be found when tracing the intersection line segments. Each patch in an object is subdivided until it is determined to be either separable or planar. Gaps are the spaces between the edges of two adjacent patches of unequal subdivision levels. The left-hand side of figure 19 shows two adjacent patches prior to subdivision. If P_1 is determined to be planar, but P_2 is not, then P_2 is subdivided. The planar representation of P_1 has an edge AB, whereas the planar representation of P_{23} has an edge CB (see right-hand side of fig. 19). These two edges are from adjacent patches but intersect P_3 at two different points. The point-ordering algorithm cannot match the points to create a continuous intersection curve.

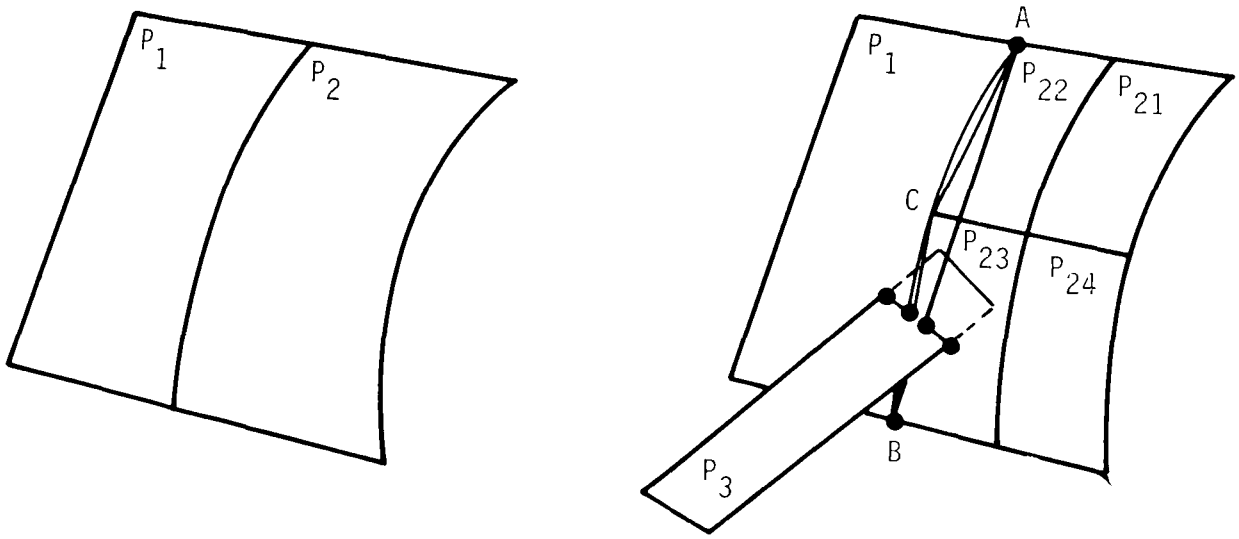


Figure 19. Formation of gaps due to unequal subdivision.

The figure shows a large gap between patches for the purpose of illustration, but in actual practice, the gap is small. The smallness of the gap allows the use of the intersection points closest to the unmatched point, which forces the match. The polygon construction then continues without failure.

Transformation From Polygon to Bezier Patch

Once the polygons in the neighborhood of the intersection are determined, they must be converted back to the Bézier representation to be consistent with the patches that were not involved. A Bézier patch has four vertices, whereas the polygons created after intersection can have any number of vertices. Therefore, to transform a polygon into a Bézier patch, the polygon must first be subdivided into quadrilaterals. In figure 20, a six-sided polygon is divided into five quadrilaterals. This division is accomplished using a trapezoidal algorithm (ref. 6). The trapezoids are then converted using the four vertices as control points. The other 12 points which determine the surface of a Bézier patch are distributed evenly along the edges and the interior of the trapezoid.

The algorithm for creating the trapezoids projects the 3-D polygon onto a 2-D plane and then rotates the polygon to avoid having two vertices on the same horizontal line. Two horizontal lines are constructed through adjacent vertices. The two points where these lines intersect the edges of the polygon and the two vertex points make up the trapezoid. This process is repeated until all the vertices have been used. The trapezoidal subdivision is described in algorithmic form as follows:

```
Procedure trapezoid
For each polygon, orderp, created by the intersection algorithm
  Begin 1
    m = 0
    nn = Number of vertex points − 1
    Is nn = 3?
      Case yes:
        poly1 = orderp1
        poly2 = orderp4
        poly3 = orderp2
        poly4 = orderp3
        Determine the 16 control points which make up a Bézier patch
        Continue with next polygon
      Case no:
        (Work only with first nn points)
```

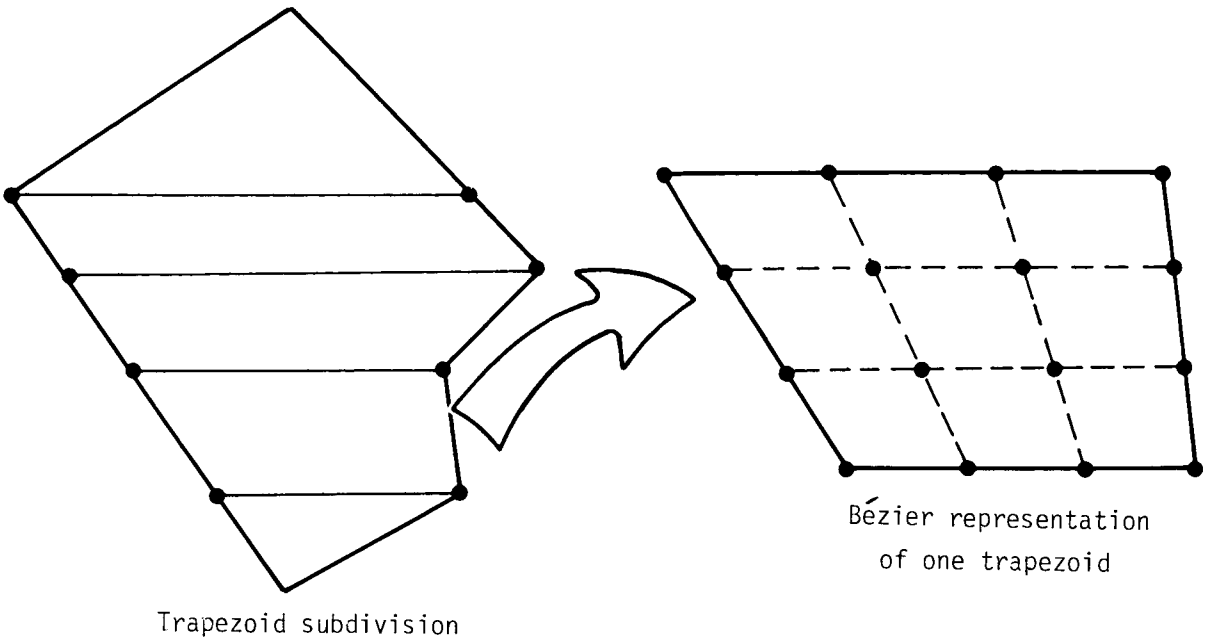


Figure 20. Conversion of n-sided polygon to Bézier patches.

Project the polygon onto the plane which produces the largest projected area (vx(i),vy(i) contains the projected 2-D vertex points, and vz(i) contains the unused 3rd coordinates)
Rotate the polygon until no 2 vy coordinates are the same
Order the coordinates ascending in vy
(vertex_pt$_1$ is the bottom segment of the first trapezoid)
sl(1,1) = vx(1)
sl(2,1) = vy(1)
sl(3,1) = vz(1)
sl(1,2) = vx(1)
sl(2,2) = vy(1)
sl(3,2) = vz(1)
(Save the 2 edges which intersect at vertex point$_i$)
Is the predecessor of vx(1) < The successor
  Case yes:
    dl(1,1) = 1
    dl(2,1 = pred(1) (pred(1) = nn)
    dl(1,2) = 1
    dl(2,2) = succ(1) (succ(1) = 2)
  Case no:
    dl(1,1) = 1
    dl(2,1) = succ(1)
    dl(1,2) = 1
    dl(2,2) = pred(1)
j = 2 (the number of edges a critical line can intersect)
For i = 2 to nn − 1 do
  Begin 2
    For k = 1 to j do
      Begin 3
        (Determine the intersection points, u(. . . , k), for y = vy(i) through edge dl(. . ., k))
          u(2,k) = vy(i)

Is vertex_pt$_i$ one of the endpoints of edge dl(..., k) (does i = dl(1,k) or dl(2,k))?
Case yes:
u(1,k) = vx(i)
u(3,k) = vx(i)
vertex(k) = True
Case no:
u(1,k) = First coordinate of the intersection point
u(3,k) = Third coordinate of the intersection point
vertex(k) = False
End 3
Are there more than 2 intersection points (is j > 2)?
Case yes:
For k = 1 to j − 1 by 2 do
Begin 4
Is vertex(k) = True?
Case yes:
Save trapezoid (tl(1,m + 1) = sl_k, tl(2,m + 1) = sl_{k+1}, tl(1,m + 2) = u_k, tl(2,m + 2) = u_{k+1})
m = m + 2
Case no:
(The intersection is not a critical segment, ignore it)
(Save the bottom segment of this trapezoid instead)
$u_k = sl_k$
$u_{k+1} = sl_{k+1}$
End 4
Case no:
Save trapezoid
m = m + 2
Evaluate case vy(i) of
Case 1: regular, vy(i) is not stalagmitic nor stalactitic
Is vy(pred(i)) > vy(succ(i))?
Case yes:
Reset the edge dl = (succ(i), i) to (i,pred(i))
Case no:
Reset the edge dl = (pred(i),i) to (i,succ(i))
Do $sl_k = u_k$ for k = 1 to j
Case 2: stalactitic, vy(i)<vy(pred(i)), vy(i)<vy(succ(i))
sl = (u_1,...,u_k,vertex_pt$_i$, vertex_pt$_i$, u_{k+1},...,u_j), where u(1,1) < u(1,k) < vx(i) < u(1,k+1) < u(1,j)
Shift the edges in dl which are located to the right of vertex_pt$_i$
Save in order the 2 edges which intersect at vertex_pt$_i$
Is vx(pred(i)) < vx(succ(i))?
Case yes:
dl(1,k+1) = i
dl(2,k+1) = pred(i)
dl(1,k+2) = i
dl(2,k+2) = succ(i)
Case no:
dl(1,k+1) = i
dl(2,k+1) = succ(i)
dl(1,k+2) = i
dl(2,k+2) = pred(i)
j = j + 2
Case 3: stalagmitic, vy(i) > vy(pred(i)), vy(i) > vy(succ(i));
sl = (u_k,...,u_{k-1},u_{k+2},...,u_j), where u(1,1)<u(1,k−1)<u(1,k+2)<u(1,j)

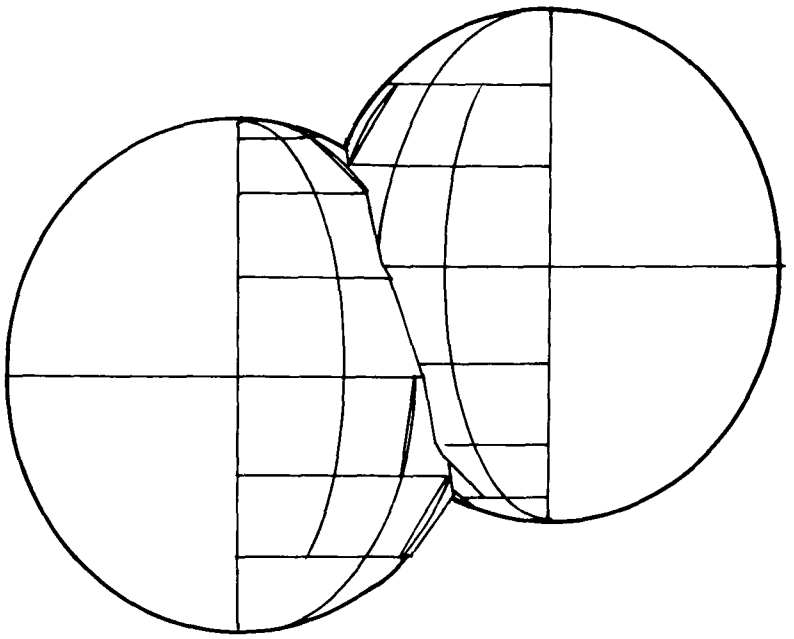


Figure 21. Union of two objects.

```
            Delete the 2 edges in dl which intersect at vertex_pt_i
            j = j - 2
        End 2
    (Vertex point_nn is the top segment of the last trapezoid)
    Save the trapezoid (tl(1,m+1) = sl_1, tl(2,m+1) = sl_2, tl(1,m+2) = vertex_pt_nn,
        tl(2,m+2) = vertex_pt_nn)
    m = m + 2
    For each trapezoid
        Begin 5
            Rotate trapezoid back to original position;
            Determine the 16 control points which make up a Bézier patch, and project them back onto the
                original polygon
        End 5
End 1
```

Combinatorial Operators

When all the intersection calculations and patch reconstructions are complete, each of the two original objects consists of a mixture of Bézier patches. Some of the patches are entirely outside the other object, and the others are entirely inside. These patches can be combined in a number of ways to form a third object.

The UNION operator creates a third object from the outside surfaces of two intersecting objects. The two objects are joined at the intersection curve or curves, and any surface from one object which lies inside the other object is not considered part of the third object. Figure 21 shows the union of two approximately spherical objects.

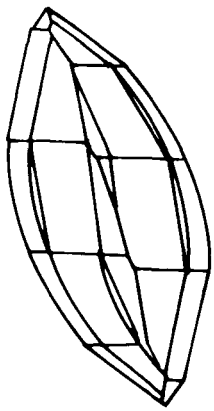


Figure 22. Intersection of two objects.

The INTERSECT operator creates a third object from the inside surfaces of two intersecting objects. Any surface from one object which lies outside the other object is not considered part of the third object. Figure 22 shows the intersection of the two previous objects.

The CUT operator creates a third object by cutting the surface of object i with the surface of object j at the intersection curve or curves and retaining sections of each object. The orientation value determines the retained sections as follows:

Orientation = 1; object 3 will be the surfaces of object j which are outside of object i;

Orientation = -1; object 3 will be the surfaces of object j which are inside of object i;

Orientation = 0; object 3 will be the surfaces of object j which are outside of object i and the surfaces of object i which are inside of object j.

Figure 23 shows examples of using the CUT operator with various combinations of parameters. The cut shown in figure 23(a) contains the portions of object 2 which are outside of object 1. This actually leaves a hole in object 2 where object 1 intersected. The cut shown in figure 23(b) consists of the portions of object 2 which are inside of object 1. Also, this result is only a piece of the surface of object 2 with a hole where object 1 intersected.

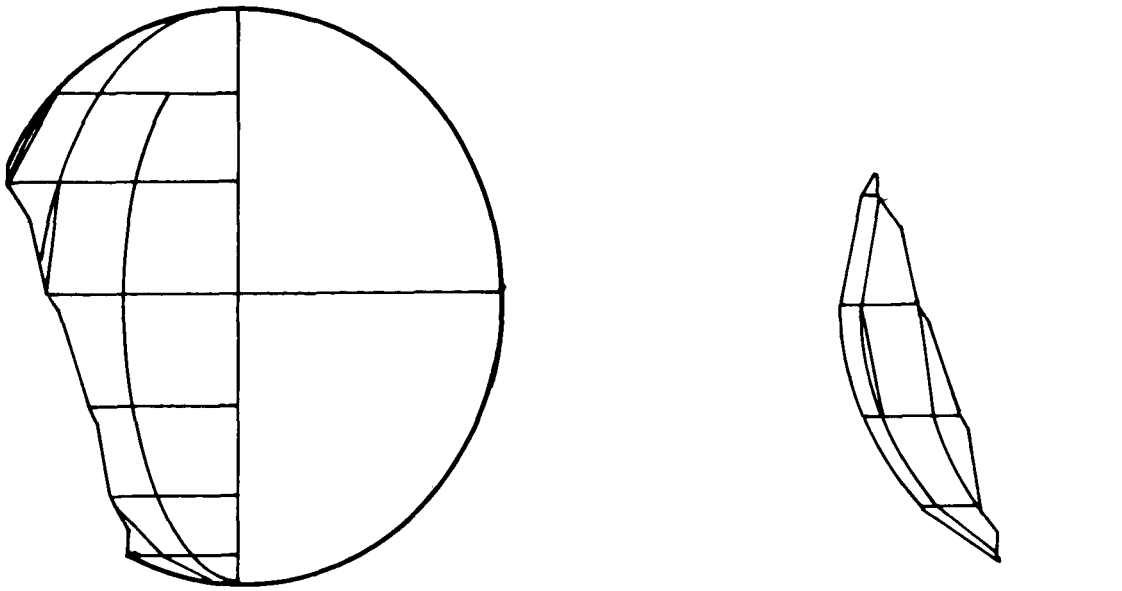
Performance Considerations

The size and execution time of the intersection program depends a great deal on the application. Because each nonseparable leaf node of one object is compared with each nonseparable leaf node of the other object, the key parameter for both size and speed is the number of leaf nodes generated for the objects. The memory space required for the program is dominated by the arrays that store leaf-node information. The two arrays described in the data structure discussion hold 60 values, mostly floating point numbers, for each node. Other arrays, which hold tree pointers and planar coefficients, add an additional ten values per node.

For the intersection illustrated in figure 21, 58 leaf nodes were generated for each object. In an actual application, a much smoother intersection is required and results in many more nodes. The time required to calculate the intersection and to construct a resultant object is roughly proportional to the product of the number of leaf nodes in one object and the number of leaf nodes in the other object. The time can vary widely, depending on the number of nodes that are actually involved in the intersection. As an example, the case shown in figure 21 took 15 seconds on a Prime 850 minicomputer. Almost all this time was spent calculating the intersection points. The subdivision and polygon reconstruction took very little time.

Concluding Remarks

Calculating the line of intersection between two three-dimensional objects and using the information



(a) $i = 1$, $j = 2$, Orientation $= 1$.

(b) $i = 1$, $j = 2$, Orientation $= -1$.

Figure 23. Examples of CUT operator.

to generate a third object is a key element in a geometry development system. This capability allows the joining of components, such as aircraft wings and fuselages. Cutouts in surfaces, such as windows, can be made. The techniques presented provide the capability to generate complex three-dimensional objects by the union or intersection of two surfaces.

The method of calculating the intersection involves subdividing the original surfaces into planar polygons and calculating intersection line segments between them. This method causes the smoothness of the surface, achieved by using bicubic patches, to be compromised in the neighborhood of the intersection. This faceting of the surface can be reduced by tightening the tolerance on the planarity of the patches at the expense of more computer time.

Further research should be conducted in this area to investigate means of maintaining surface smoothness. Another area of research should be the blending of two objects rather than a straight intersection. Slope information provided by the patch definition could be used to generate fillets or blending patches between objects.

NASA Langley Research Center
Hampton, VA 23665
April 2, 1985

References

1. Wilhite, A. W.; and Rehder, J. J.: AVID: A Design System for Technology Studies of Advanced Transportation Concepts. AIAA Paper 79-0872, May 1979.
2. Carlson, Wayne Earl: Techniques for the Generation of Three Dimensional Data for Use in Complex Image Synthesis. Ph.D. Diss. (NSF Grant No. MCS 79-23670), Ohio State Univ., Sept. 1982.
3. Lane, Jeffrey M.; and Riesenfeld, Richard F.: A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces. *IEEE Trans. Pattern Anal. & Mach. Intell.*, vol. PAMI-2, no. 1, Jan. 1980, pp. 35–46.

4. Foley, James D.; and Van Dam Andries: *Fundamentals of Interactive Computer Graphics*. Addison-Wesley Pub. Co., 1982.
5. Bézier, P.: Mathematical and Practical Possibilities of UNISURF. *Computer Aided Geometric Design*, Robert E. Barnhill and Richard F. Riesenfeld, eds., Academic Press, Inc., 1974, pp. 127–152.
6. Lee, D. T.: Shading of Regions on Vector Display Devises. *Comput. Graphics*, vol. 15, no. 3, Aug. 1981, pp. 37–44.

1. Report No. NASA TP-2454	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Intersection of Three-Dimensional Geometric Surfaces		5. Report Date July 1985
		6. Performing Organization Code 506-63-23-02
7. Author(s) Vicki K. Crisp, John J. Rehder, and James L. Schwing		8. Performing Organization Report No. L-15911
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665		10. Work Unit No.
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Technical Paper
		14. Sponsoring Agency Code

15. Supplementary Notes

Vicki K. Crisp: Kentron International, Inc., Hampton, Virginia.
John J. Rehder: Langley Research Center, Hampton, Virginia.
James L. Schwing: Old Dominion University, Norfolk, Virginia.

16. Abstract

Calculating the line of intersection between two three-dimensional objects and using the information to generate a third object is a key element in a geometry development system. Techniques are presented for the generation of three-dimensional objects, the calculation of a line of intersection between two objects, and the construction of a resultant third object. The objects are closed surfaces consisting of adjacent bicubic parametric patches using Bézier basis functions. The intersection determination involves subdividing the patches that make up the objects until they are approximately planar and then calculating the intersection between planes. The resulting straight-line segments are connected to form the curve of intersection. The polygons in the neighborhood of the intersection are reconstructed and put back into the Bézier representation. A third object can be generated using various combinations of the original two. Several examples are presented. Special cases and problems were encountered, and the method for handling them is discussed. The special cases and problems included intersection of patch edges, gaps between adjacent patches because of unequal subdivision, holes, or islands within patches, and computer round-off error.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
Geometry modelling Intersection Three-dimensional surfaces Bézier surfaces Bicubic surfaces Computer graphics Computer-aided design	Unclassified--Unlimited Subject Category 59

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	31	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1985